Kostenloses eBook

LERNEN testng

Free unaffiliated eBook created from
Stack Overflow contributors.

#testng

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: testng

It is an unofficial and free testng ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official testng.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit testng

Bemerkungen

In diesem Abschnitt erhalten Sie einen Überblick darüber, was testng ist und warum ein Entwickler es möglicherweise verwenden möchte.

Es sollte auch alle großen Themen in testng erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für testng neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Versionen

Ausführung	Datum
1,0	2017-06-07

Examples

Installation oder Setup

TestNG erfordert die Verwendung von JDK 7 oder höher.

Gemäß http://testng.org/doc/download.html müssen Sie zur Installation von testng die Abhängigkeit von testng zu Ihrer maven pom.xml- oder gradle-Datei build.gradle hinzufügen

Maven:

```
<repositories>
  <repository>
    <id>jcenter</id>
    <name>bintray</name>
    <url>http://jcenter.bintray.com</url>
  </repository>
</repositories>

<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>6.9.12</version>
  <scope>test</scope>
</dependency>
```

Gradle:

```
repositories {
    jcenter()
```

```
}

dependencies {
    testCompile 'org.testng:testng:6.9.12'
}
```

Weitere Optionen finden Sie auf der offiziellen Seite .

Schnellprogramm mit TestNG

```
package example;

import org.testng.annotations.*; // using TestNG annotations

public class Test {

 @BeforeClass
 public void setUp() {
   // code that will be invoked when this test is instantiated
 }

 @Test(groups = { "fast" })
 public void aFastTest() {
   System.out.println("Fast test");
 }

 @Test(groups = { "slow" })
 public void aSlowTest() {
    System.out.println("Slow test");
 }

}
```

Die Methode `setUp()` wird aufgerufen, nachdem die `setUp()` wurde und bevor eine Testmethode ausgeführt wird. In diesem Beispiel werden wir die Gruppe schnell `aFastTest()` , sodass `aFastTest()` aufgerufen wird, während `aSlowTest()` übersprungen wird.

TestNG Hello World Beispiel

Das Schreiben und Ausführen eines einfachen `TestNG` Programms besteht hauptsächlich aus drei Schritten.

1. Code - Schreiben Sie die Geschäftslogik Ihres Tests und kommentieren Sie sie mit TestNG-Anmerkungen
2. Konfigurieren - fügen Sie Informationen zu Ihrem Test in `testng.xml` oder in `build.xml`
3. TestNG ausführen - kann von der Befehlszeile, ANT, IDE wie Eclipse, IntelliJs IDEA aufgerufen werden.

Kurze Erläuterung des Beispiels (was getestet werden muss) :

Wir haben eine `RandomNumberGenerator` Klasse, die über die Methode `generateFourDigitPin` , die eine vierstellige PIN generiert und als `int` zurückgibt. Wir wollen hier also testen, ob diese Zufallszahl aus 4 Ziffern besteht oder nicht. Unten ist der Code:

Zu testende Klasse :

```
package example.helloworld;

public class RandomNumberGenerator {

public int generateFourDigitPin(){
    return (int)(Math.random() * 10000);
}
}
```

Die TestNG-Testklasse :

```
package example.helloworld;

import org.testng.Assert;
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class TestRandomNumberGenerator {

    RandomNumberGenerator rng = null;

    @BeforeClass
    public void deSetup(){
        rng = new RandomNumberGenerator();
    }

    @Test
    public void testGenerateFourDigitPin(){
        int randomNumber = rng.generateFourDigitPin();
        Assert.assertEquals(4, String.valueOf(randomNumber).length());
    }

    @AfterClass
    public void doCleanup(){
        //cleanup stuff goes here
    }
}
```

Ther testng.xml :

```
<suite name="Hello World">
    <test name="Random Number Generator Test">
        <classes>
            <class name="example.helloworld.TestRandomNumberGenerator" />
        </classes>
    </test>
</suite>
```

Führen Sie die TestNG-Suite mit Gradle aus

Beispieldatei `build.gradle` :

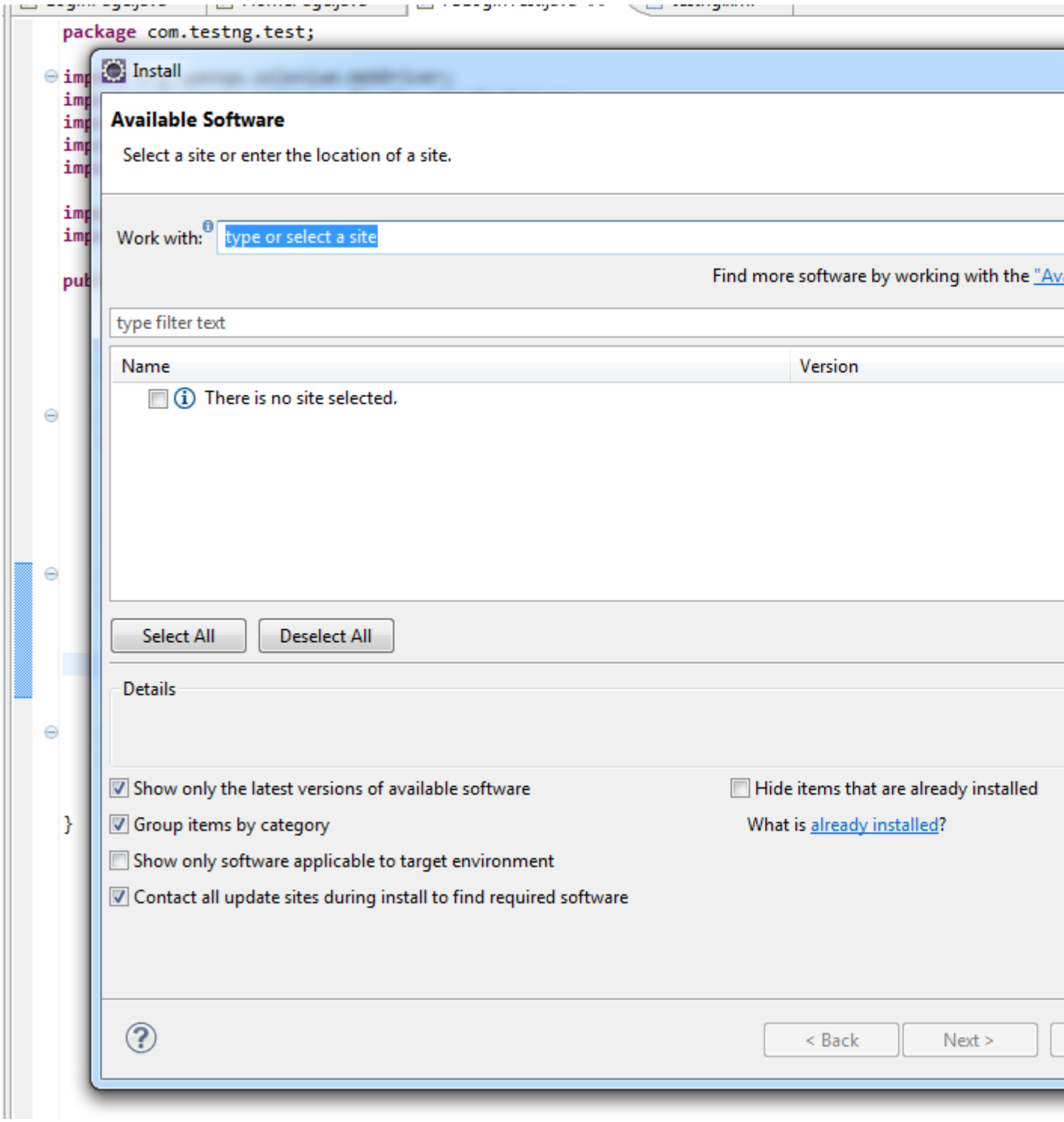
```
plugin: 'java'
```

```
repositories {
    mavenLocal()
    mavenCentral()
    jcenter()
}

dependencies {
    compile "org.testng:testng:6.9.12"
}

test {
    useTestNG() {
    suiteXmlBuilder().suite(name: 'Sample Suite') {
        test(name : 'Sample Test') {
            classes('') {
                'class'(name: 'your.sample.TestClass')
            }
        }
    }
}
```

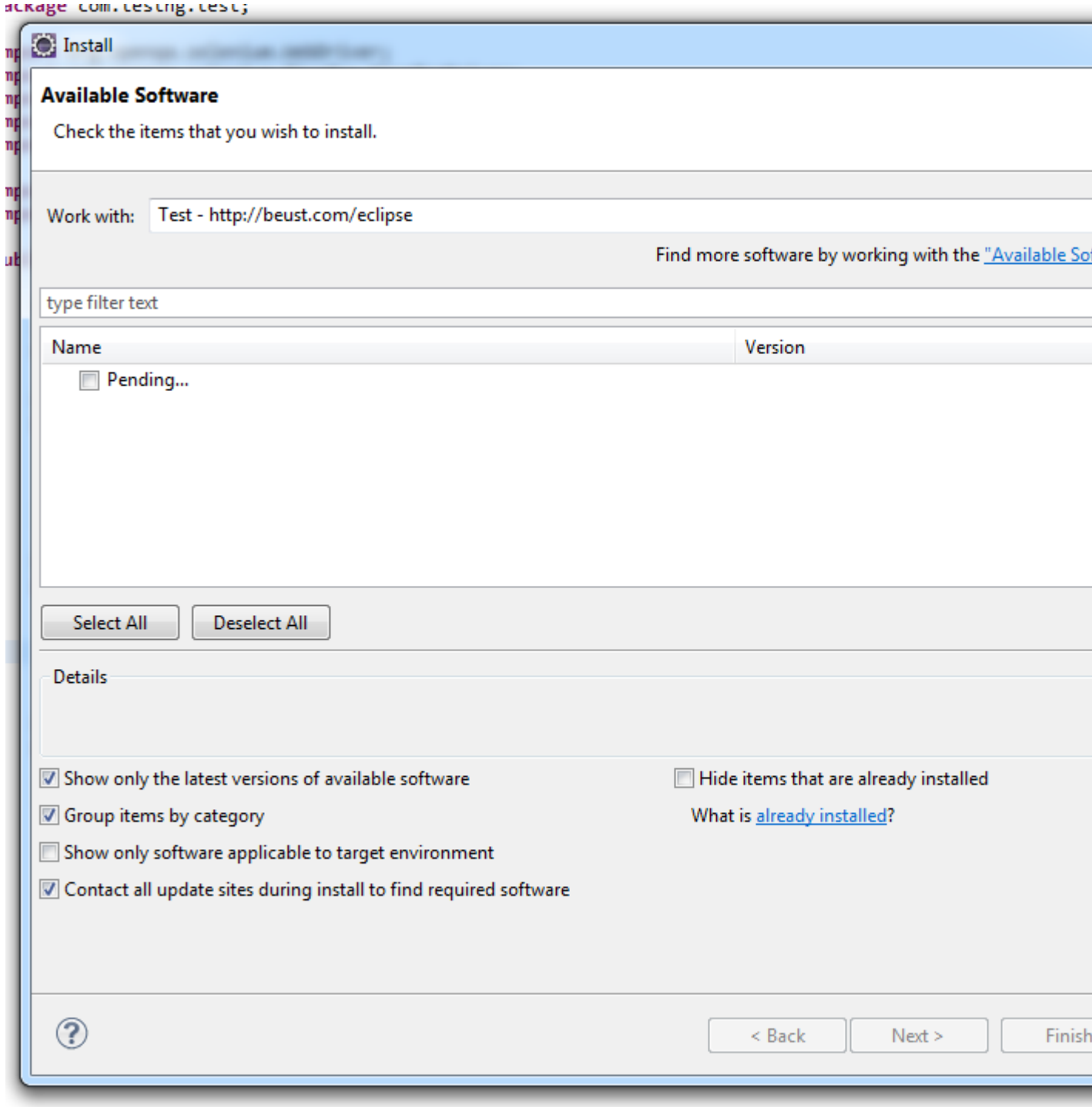
So konfigurieren Sie TestNG in Eclipse & Run test mit xml

So installieren Sie TestNG in Eclipse

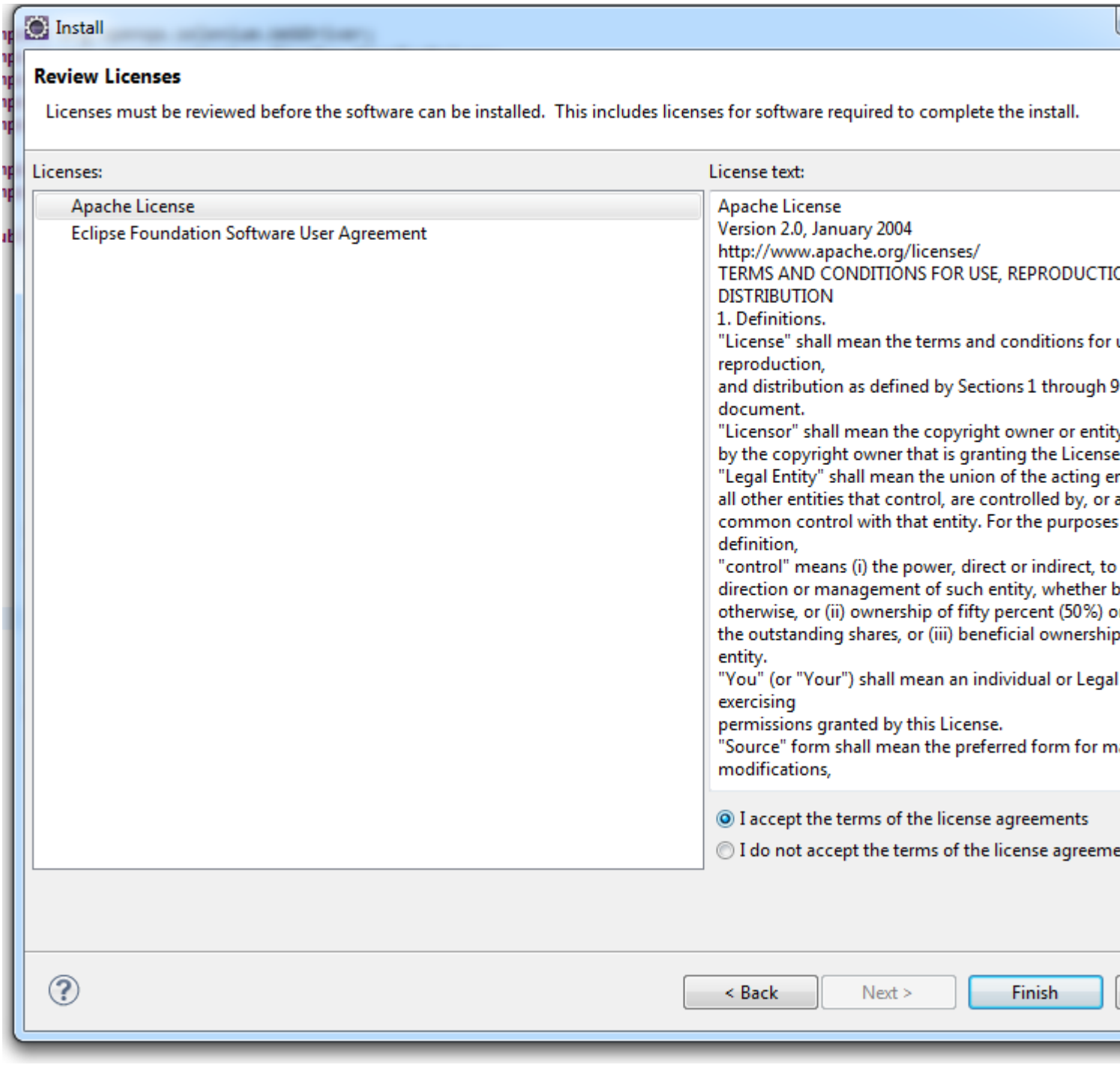
1. Eclipse öffnen
2. Klicken Sie auf Hilfe> Neue Software installieren



3. Klicken Sie auf Hinzufügen
4. Geben Sie Name und URL an - http://beust.com/eclipse



5. Wählen Sie TestNG aus
6. Weiter klicken



7. Klicken Sie auf Fertig stellen
8. Die Installation von TestNG wird einige Zeit dauern

Einmal installiert, starten Sie Eclipse neu.

Lassen Sie uns ein TestNG-Projekt erstellen

1. Datei> Neu> Java-Projekt> Geben Sie einen Namen ein und klicken Sie auf Fertig stellen

2. Erstellen Sie eine Klasse als TestNGClass

3. Erstellen Sie folgende Klasse

    1.LoginPage.class

2.HomePage.class

3.FBLoginTest.class

Hier ist der Code:

LoginPage-Klasse

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {

    @FindBy(id = "email")
    private WebElement username;

    @FindBy(id = "pass")
    private WebElement password;

    @FindBy(xpath = ".//input[@data-testid='royal_login_button']")
    private WebElement login;

    WebDriver driver;

    public LoginPage(WebDriver driver){
        this.driver = driver;
         PageFactory.initElements(driver, this);
    }
    public void enterUserName(String name){
        username.clear();
        username.sendKeys(name);
    }

    public void enterPassword(String passwrd){
        password.clear();
        password.sendKeys(passwrd);
    }


    public HomePage clickLoginButton(){
        login.click();
        return new HomePage(driver);
    }
}
```

HomePage-Klasse .

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class HomePage {

    @FindBy(id = "userNavigationLabel")
    private WebElement userDropdown;
```

```
    WebDriver driver;

    public HomePage(WebDriver driver){
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    public boolean isUserLoggedIn(){
        return userDropdown.isDisplayed();
    }

}
```

FBLoginTest-Klasse

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.AfterClass;

import com.testng.pages.HomePage;
import com.testng.pages.LoginPage;

public class FBLoginTest {

    WebDriver driver;
    LoginPage loginPage;
    HomePage homePage;

    @BeforeClass
    public void openFBPage(){
        driver = new FirefoxDriver();
        driver.get("https://www.facebook.com/");
        loginPage = new LoginPage(driver);
    }

    @Test
    public void loginToFB(){
        loginPage.enterUserName("");
        loginPage.enterPassword("");
        homePage = loginPage.clickLoginButton();
        Assert.assertTrue(homePage.isUserLoggedIn());
    }

    @AfterClass
    public void closeBrowser(){
        driver.quit();
    }

}
```

Hier kommt die testng-XML: Klicken Sie mit der rechten Maustaste auf Projekt, erstellen Sie eine XML-Datei und kopieren Sie diesen Inhalt.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<suite name="Suite">
  <test name="Test">
    <classes>
      <class name="com.testng.FBLoginTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

So fügen Sie Selen-Standalone-Glas hinzu:
Laden Sie den neuesten Standalone-Selenbehälter herunter und fügen Sie ihn im Build-Pfad des Projekts hinzu.

1. Klicken Sie mit der rechten Maustaste auf Projekt> Erstellungspfad> Erstellungspfad konfigurieren> Bibliotheken auswählen> Externe Jars hinzufügen

Wie führe ich die TestNG-XML aus? Klicken Sie mit der rechten Maustaste auf xml> Ausführen als> TestNGSuite

Viel Spaß beim Codieren :)

Erste Schritte mit testng online lesen: https://riptutorial.com/de/testng/topic/5393/erste-schritte-mit-testng

Kapitel 2: @Test-Anmerkung

Syntax

- @Prüfung
- @Test (Attribut1 = Attributwert, Attributwert = Attributwert usw.)

Parameter

Parameter	Einzelheiten
alwaysRun	Bei true wird diese Testmethode immer ausgeführt, auch wenn sie von einer fehlgeschlagenen Methode abhängig ist.
Datenanbieter	Der Name des Datenanbieters für diese Testmethode.
dataProviderClass	Die Klasse, in der nach dem Datenanbieter gesucht werden soll. Wenn nicht angegeben, wird der Datenprovider nach der Klasse der aktuellen Testmethode oder einer ihrer Basisklassen durchsucht. Wenn dieses Attribut angegeben ist, muss die Datenanbietermethode für die angegebene Klasse statisch sein.
Abhängig vonOnGroups	Die Liste der Gruppen, von denen diese Methode abhängig ist.
hängt vonOnMethods ab	Die Liste der Methoden, von denen diese Methode abhängig ist.
Beschreibung	Die Beschreibung für diese Methode.
aktiviert	Gibt an, ob Methoden für diese Klasse / Methode aktiviert sind.
erwarteteAusnahmen	Die Liste der Ausnahmen, die eine Testmethode auslösen soll. Wenn keine Ausnahme oder eine andere als eine in dieser Liste geworfen wird, wird dieser Test als fehlerhaft markiert.
Gruppen	Die Liste der Gruppen, zu denen diese Klasse / Methode gehört.
invocationCount	Die Häufigkeit, mit der diese Methode aufgerufen werden soll.
invocationTimeOut	Die maximale Anzahl von Millisekunden, die dieser Test für die kumulierte Zeit aller Aufrufzähler benötigt. Dieses Attribut wird ignoriert, wenn invocationCount nicht angegeben ist.
Priorität	Die Priorität für diese Testmethode. Niedrigere Prioritäten werden zuerst festgelegt.

Parameter	Einzelheiten
erfolg prozentsatz	Der von dieser Methode erwartete Prozentsatz des Erfolgs
singleThreaded	Wenn true festgelegt ist, werden alle Methoden in dieser Testklasse garantiert im selben Thread ausgeführt, auch wenn die Tests derzeit mit `parallel="methods"` . Dieses Attribut kann nur auf Klassenebene verwendet werden und wird bei Verwendung auf Methodenebene ignoriert. **Hinweis** : Dieses Attribut wurde früher als sequenziell bezeichnet (jetzt nicht mehr unterstützt).
Auszeit	Die maximale Anzahl von Millisekunden, die dieser Test dauert.
threadPoolSize	Die Größe des Thread-Pools für diese Methode. Die Methode wird von mehreren Threads aufgerufen, wie von invocationCount angegeben. **Hinweis** : Dieses Attribut wird ignoriert, wenn invocationCount nicht angegeben ist

Examples

Schnelles Beispiel für die @ Test-Annotation

`@Test` Annotation kann auf jede **Klasse** oder **Methode** angewendet **werden** . Diese Annotation kennzeichnet eine Klasse oder eine Methode als Teil des Tests.

1. `@Test` auf Methodenebene - Annotierte Methode als Testmethode markieren
2. `@Test` auf Klassenebene
    - Die `@Test` Annotation auf Klassenebene `@Test` , dass alle öffentlichen Methoden der Klasse zu Testmethoden werden, auch wenn sie nicht kommentiert werden.
    - `@Test` Annotation kann auch für eine Methode wiederholt werden, wenn Sie bestimmte Attribute hinzufügen möchten.

Beispiel für `@Test` auf Methodenebene :

```
import org.testng.annotations.Test;

public class TestClass1 {
    public void notTestMethod() {
    }

    @Test
    public void testMethod() {
    }
}
```

Beispiel für `@Test` auf Klassenebene :

```
import org.testng.annotations.Test;

@Test
```

```
public class TestClass2 {
    public void testMethod1() {
    }

    @Test
    public void testMethod2() {
    }
}
```

@Test-Anmerkung online lesen: https://riptutorial.com/de/testng/topic/6716/-test-anmerkung

Kapitel 3: Parametrisierte Tests

Examples

Datenanbieter

Datenanbieter ermöglichen das Erstellen mehrerer Testeingaben, die innerhalb eines Tests ausgeführt werden. Betrachten wir einen Test, mit dem sichergestellt wird, dass die Zahlen korrekt verdoppelt werden. Erzeugen von Daten - Provider eine statische Methode bereitzustellen , die entweder kehrt `Object[][]` oder `Iterator<Object[]>` (letztere ermöglicht lazy Berechnung der Testeingänge) mit annotierten `@DataProvider` Annotation, mit Eigenschaft `name` eine eindeutige Zeichenfolge identifiziert das Wesen Anbieter.

```
import org.testng.annotations.DataProvider;

public class DoublingDataProvider {
    public final static String DOUBLING_DATA_PROVIDER = "doublingDataProvider";

    @DataProvider(name = DOUBLING_DATA_PROVIDER)
    public static Object[][] doubling() {
        return new Object[][]{
                new Object[]{1, 2},
                new Object[]{2, 4},
                new Object[]{3, 6}
        };
    }
}
```

Im obigen Fall stellt jedes `Object[]` einen Datensatz für einen einzelnen Testfall dar - hier die zu verdoppelende Anzahl, gefolgt vom erwarteten Wert nach der Verdoppelung.

Um den Datenprovider zu verwenden, geben Sie die `dataProvider` Eigenschaft des Tests mit dem Namen des Providers an. Wenn die Providermethode außerhalb der `dataProviderClass` oder ihrer Basisklassen definiert wurde, müssen Sie auch die Eigenschaft `dataProviderClass` angeben. Die Testmethode sollte Parameter annehmen, die den Elementen der Testfallbeschreibung entsprechen - hier sind es zwei Punkte.

```
import org.testng.annotations.Test;

import static org.testng.Assert.assertEquals;

public class DoublingTest {

    @Test(dataProvider = DoublingDataProvider.DOUBLING_DATA_PROVIDER, dataProviderClass =
DoublingDataProvider.class)
    public void testDoubling(int number, int expectedResult) {
        assertEquals(number * 2, expectedResult);
    }
}
```

Parametrisierte Tests online lesen: https://riptutorial.com/de/testng/topic/5684/parametrisierte-

tests

Kapitel 4: TestNG - Ausführungsverfahren

Examples

Ausführungsprozedur der TestNG-Test-API-Methoden

```
public class TestngAnnotation {
   // test case 1
   @Test
   public void testCase1() {
      System.out.println("in test case 1");
   }

   // test case 2
   @Test
   public void testCase2() {
      System.out.println("in test case 2");
   }

   @BeforeMethod
   public void beforeMethod() {
      System.out.println("in beforeMethod");
   }

   @AfterMethod
   public void afterMethod() {
      System.out.println("in afterMethod");
   }

   @BeforeClass
   public void beforeClass() {
      System.out.println("in beforeClass");
   }

   @AfterClass
   public void afterClass() {
      System.out.println("in afterClass");
   }

   @BeforeTest
   public void beforeTest() {
      System.out.println("in beforeTest");
   }

   @AfterTest
   public void afterTest() {
      System.out.println("in afterTest");
   }

   @BeforeSuite
   public void beforeSuite() {
      System.out.println("in beforeSuite");
   }

   @AfterSuite
   public void afterSuite() {
      System.out.println("in afterSuite");
```

```
   }

}
```

Lassen Sie uns in C:> WORKSPACE die Datei testng.xml erstellen, um Anmerkungen auszuführen.

```
<suite name="Suite1">
  <test name="test1">
    <classes>
       <class name="TestngAnnotation"/>
    </classes>
  </test>
</suite>
```

C: \ WORKSPACE> javac TestngAnnotation.java

Führen Sie nun die Datei testng.xml aus, in der der in der bereitgestellten Testfallklasse definierte Testfall ausgeführt wird.

```
in beforeSuite
in beforeTest
in beforeClass
in beforeMethod
in test case 1
in afterMethod
in beforeMethod
in test case 2
in afterMethod
in afterClass
in afterTest
in afterSuite

===============================================
Suite
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

Das Ausführungsverfahren ist wie folgt:

1. **Vorher** wird die **beforeSuite ()** -Methode nur einmal ausgeführt.
2. Schließlich wird die **afterSuite ()** -Methode nur einmal ausgeführt.
3. Sogar die Methoden **beforeTest ()** , **beforeClass ()** , **afterClass ()** und **afterTest ()** werden nur einmal ausgeführt.
4. **Die beforeMethod ()** -Methode wird für jeden Testfall ausgeführt, jedoch bevor der Testfall ausgeführt wird.
5. **Die afterMethod ()** -Methode wird für jeden Testfall ausgeführt, jedoch nach Ausführung des Testfalls.
6. Zwischen **beforeMethod ()** und **afterMethod ()** wird jeder Testfall ausgeführt.

TestNG - Ausführungsverfahren online lesen: https://riptutorial.com/de/testng/topic/7889/testng---ausfuhrungsverfahren

Kapitel 5: TestNG-Gruppen

Syntax

- @Test (groups = {"group1", "group.regression"}, hängt vonOnGroups = {"group2", "group3"})

Examples

TestNG Gruppiert Konfiguration und grundlegendes Beispiel

Gruppen können unter `Suite` und / oder `Test` Element von `testng.xml` . Alle Gruppen, die als in `tesng.xml` enthalten `tesng.xml` sind, werden für die Ausführung berücksichtigt. Ausgeschlossene Gruppen werden ignoriert. Wenn eine `@Test` Methode mehrere Gruppen hat und aus diesen Gruppen Einzelgruppen in `testng.xml` ausgeschlossen sind, wird die `testng.xml` `@Test` Methode nicht ausgeführt.

Unten ist die typische `testng.xml` Konfiguration auf `Test` - Ebene Gruppen für den Betrieb:

```
<suite name="Suite World">
<test name="Test Name">
  <groups>
    <run>
      <include name="functest" />
      <exclude name="regtest" />
    </run>
  </groups>
  <classes>
    <class name="example.group.GroupTest"/>
  </classes>
</test>
</suite>
```

Und so wird es in der Testklasse aussehen:

```
package example.group;

import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;

public class GroupTest {

    @BeforeClass
    public void deSetup(){
        //do configuration stuff here
    }

    @Test(groups = { "functest", "regtest" })
    public void testMethod1() {
    }
```

```
    @Test(groups = {"functest", "regtest"} )
    public void testMethod2() {
    }

    @Test(groups = { "functest" })
    public void testMethod3() {
    }

    @AfterClass
    public void cleanUp(){
        //do resource release and cleanup stuff here
    }
}
```

Beim Ausführen dieser `GroupTest TestNG` Klasse wird nur `testMethod3()` ausgeführt.

Erläuterung:

- `<include name="functest" />` Alle Testmethoden der `functest` group können ausgeführt werden, wenn sie von keiner anderen Gruppe ausgeschlossen werden.
- `<exclude name="regtest" />` keine Testmethoden der `regtest` Gruppe ausgeführt werden.
- `testMethod1()` und `testMethod2()` befinden sich in der `regtest` Gruppe, sodass sie nicht ausgeführt wurden.
- `testMethod3()` befindet sich in der Gruppe " `regtest` " und wird daher ausgeführt.

TestNG-Metagruppen - Gruppen von Gruppen

Mit TestNG können Gruppen definiert werden, die andere Gruppen enthalten können. MetaGroups kombinieren logisch eine oder mehrere Gruppen und steuern die Ausführung der zu diesen Gruppen gehörenden `@Test` Methoden.

Im folgenden Beispiel gibt es verschiedene `@Test` Methoden, die zu verschiedenen Gruppen gehören. Wenige sind spezifisch für bestimmte Stapel und wenige sind Regressions- und Akzeptanztests. Hier können Metagruppen angelegt werden. Lassen Sie uns zwei einfache **MetaGruppen auswählen** :

1. `allstack` - enthält sowohl die Gruppen `liux.jboss.oracle` als auch `aix.was.db2` und ermöglicht das `aix.was.db2` aller Testmethoden, die zu einer dieser Gruppen gehören.
2. `systemtest` - umfasst alle `allstack` , `regression` und `acceptance` und ermöglicht das `allstack` aller Testmethoden, die zu einer dieser Gruppen gehören.

testng.xml Konfiguration

```
<suite name="Groups of Groups">
    <test name="MetaGroups Test">
        <groups>
            <!-- allstack group includes both liux.jboss.oracle and aix.was.db2 groups -->
            <define name="allstack">
                <include name="liux.jboss.oracle" />
                <include name="aix.was.db2" />
            </define>

            <!-- systemtest group includes all groups allstack, regression and acceptance -->
```

```
            <define name="systemtest">
                <include name="allstack" />
                <include name="regression" />
                <include name="acceptance" />
            </define>

            <run>
                <include name="systemtest" />
            </run>
        </groups>

        <classes>
            <class name="example.group.MetaGroupsTest" />
        </classes>
    </test>

</suite>
```

MetaGroupsTest- Klasse

```
package example.group;

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class MetaGroupsTest {

    @BeforeMethod
    public void beforeMethod(){
        //before method stuffs - setup
    }

    @Test(groups = { "liux.jboss.oracle", "acceptance" })
    public void testOnLinuxJbossOracleStack() {
        //your test logic goes here
    }

    @Test(groups = {"aix.was.db2", "regression"} )
    public void testOnAixWasDb2Stack() {
        //your test logic goes here
    }

    @Test(groups = "acceptance")
    public void testAcceptance() {
        //your test logic goes here
    }

    @Test(groups = "regression")
    public void testRegression(){
        //your test logic goes here
    }

    @AfterMethod
    public void afterMthod(){
        //after method stuffs - cleanup
    }
}
```

TestNG-Gruppen online lesen: https://riptutorial.com/de/testng/topic/5821/testng-gruppen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit testng	Atul Dwivedi, Community, Idos, mackowski, RocketRaccoon, Sudha Velan
2	@Test-Anmerkung	Atul Dwivedi, Benoit
3	Parametrisierte Tests	Benoit, mszymborski
4	TestNG - Ausführungsverfahren	Shrikant
5	TestNG-Gruppen	Atul Dwivedi